SHARP APL Release 20.0

Guide for APL Programmers

I.P. Sharp Associates Limited, a Reuter Company

SHARP APL is a registered trademark of I.P. Sharp Associates Limited, a Reuter Company.

A publication of:

I.P. Sharp Associates Limited, a Reuter Company
Software Technology Division
2 First Canadian Place, Suite 1900
Toronto, Ontario
Canada M5X 1E3

Printed in Canada, September 1989
© I.P. Sharp Associates Limited, a Reuter Company 1989

All rights reserved. Reproduction in whole or part is prohibited without the written consent of the copyright owner.

The information contained in this publication is accurate to the best of the company's knowledge. However, I.P. Sharp Associates Limited disclaims any liability resulting from the use of this information and reserves the right to make changes without notice.

Publication code: 1047-8909-E20

Abstract

This manual describes all changes to SHARP APL since Release 19.0 as they relate to programmers. It should be read by all programmers using SHARP APL (professional programmers as well as occasional interactive users). It is designed to be used with the *SHARP APL Reference Manual* and the *Guide for APL Programmers* for previous releases of SHARP APL; these books provide a complete description of what SHARP APL is and how best to use it.

This manual and the *Internal and Operational Changes* manual describe all of the changes to SHARP APL for Version 20.

Related Publications

The following publications provide information that may be of use when programming with SHARP APL. Copies of these publications can be obtained from your I.P. Sharp representative.

Function Monitor Facility User Guide, publication code 0749-8711-E1
Internal and Operational Changes, publication code 1049-8909-E20
SHARP APL Reference Manual, publication code 79RM05
Transaction Reporting Facility Programmer's Guide, publication code 1054-8909-E20
"A Dictionary of APL," Kenneth E. Iverson, in *APL Quote Quad*, volume 18, number 1, September 1987

Conventions Used in This Manual

For all examples of input and output used in this manual, the index origin system variable, `⎕io`, is set to zero.

Examples used in this manual have the position and spacing system variable, `⎕ps`, set to `¯1 ¯1 0 1` unless specified otherwise.

Examples of input to (and output from) SHARP APL use the APL character set. (You can display the APL character set using the system function `⎕av`.) In the APL character set, the letters A through Z occur three times. The examples in this manual use the first alphabet, `⎕av[86+⍳26]`. When necessary, second

alphabet characters, `⎕av[113+⍳26]`, are used. Third alphabet characters (`⎕av[166+⍳26]`) are not used in this manual.

First alphabet characters are represented in this manual as follows. These characters are obtained by typing in the base (unshifted) keyboard.

```
      ⎕av[86+⍳26]
abcdefghijklmnopqrstuvwxyz
```

Second alphabet characters are represented as follows. These characters are obtained by specifying the alternate character set (for example, by holding down the **Alt** key) and typing the desired characters.

```
      ⎕av[113+⍳26]
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Due to the wide variety of input and output devices that can be used with SHARP APL, the first and second alphabets used in the examples may not be the same as those that appear on your terminal or printer. To determine what the first and second alphabets look like on your terminal (or printer), display (or print) the appropriate locations in `⎕av`.

CONTENTS

PREFACE
- Abstract iii
- Related Publications iii
- Conventions Used in This Manual iii

CHAPTER 1. INTRODUCTION

CHAPTER 2. LANGUAGE ENHANCEMENTS
- In (⍷) Added 5
- Nubsieve (≠) Added 6
- Format (⍕) Modified 7
- Raze (↓) Added 7
- Disclose (>) Modified 9

CHAPTER 3. FILE SYSTEM ENHANCEMENTS
- □*fhold* Modified 11
- Library Access Controls Added 12
  - □*create* 14
  - □*rename* 14
  - □*lib* 14
  - □*rdacl* 14
  - □*stacl* 14
- □*create* Modified 15
- □*rdfi* Added 15

CHAPTER 4. SYSTEM ENHANCEMENTS
- Transaction Reporting Facility Added 17
- Function Monitor Facility (□*fm*) Added 17
- Termination Workspace ID (□*twsid*) Added 18
- 2 □*ws* 3 Modified 19
- □*sc* Post at Shutdown 20
- 6 □*ws* Modified 20
- Changes to Shared Variable Functions 20
  - Monadic □*svn* Modified 21
  - Dyadic □*svn* Added 22
  - □*svq* Modified 22
  - □*svo* Modified 23

CHAPTER 5. OTHER ENHANCEMENTS
- Fix Functions □*fx* and 3 □*fd* Modified 25
- System Function Editor Modified 25

) *opr* and) *oprn* Messages to System Console 26
Modified Character Control for Non-APL Asynchronous Terminals 26
Non-APL Asynchronous Character Control Function 27

CHAPTER 6. FUNCTIONS AND PRODUCTS NO LONGER SUPPORTED
APL MPX Interface (AMPX) Replaced 29
Terminal Types That Are No Longer Supported 29

INDEX 31

CHAPTER 1. INTRODUCTION TO SHARP APL RELEASE 20

SHARP APL Version 20 is the product of changes to the base system and to the file system, along with any corresponding changes necessary to other parts of the system. These changes are designed to help your data center meet the growth needs of your APL user community. Version 20 includes system enhancements and improved capabilities that can help your data center provide efficient, dependable service to a large volume of concurrent APL users.

Language Enhancements

The following new and changed functions in the SHARP APL language are included in Version 20:

In is now available to assist in pattern-matching searches.

Nubsieve is now available to determine unique major cells in numeric, character, and boxed data.

19.8 **Format** is changed so that the result of a dyadic format with an empty vector for a left argument is now identical to the result of a monadic format.

Raze is now available to reconstitute arrays of data by opening the elements of the array and assembling the opened elements along the leading axis.

Disclose is changed to permit a permissive treatment of frame building. Disclose now automatically pads your data so that you can mix data of differing ranks and shapes, making boxed data simpler and more intuitive to use.

File System Enhancements

The following new and changed functions make the SHARP APL file system more efficient and easier to use.

19.8 □*fhold* is changed so that you can hold a range of file components rather than the entire file. This permits several applications to hold the same file simultaneously as long as the ranges held do not overlap.

Library access controls are available to permit you to grant access to entire libraries, rather than on a file-by-file basis. These controls govern which accounts can create, rename, and list files in your library, as well as who can read or set the library access controls.

`□create` is changed so that you can specify a starting component number when you create a file. This makes it easier to match component ranges of files imported from other systems.

`□rdfi` is now available to report the account number and time stamp information associated with the creation of a specified file, the latest setting of its access matrix, and its latest alteration.

System Enhancements

The following new and changed functions are available to set or report on system information. These enhancements can help you efficiently use the services and resources available with SHARP APL.

The **Transaction Reporting Facility** is available to collect information on the use of SHARP APL. The information collected is stored in a system file and can be used as the basis for applications such as determining usage statistics, application performance analysis, or customer billing systems.

19.8 `□fm`, the function monitor facility, is now available to measure and report on CPU time and elapsed time used during the execution of user-defined functions. This facility can be used for monitoring entire functions or specific lines within functions.

`□twsid`, the termination workspace ID function, is now available so that you can specify whether the current workspace is to be saved in the event of the abnormal termination of a task. It also allows you to specify a termination workspace ID (other than the default, `continue`) to be used for the saved workspace.

19.8 `2 □ws 3`, the workspace reporting function, is changed so that it can now provide information on the pending shutdown of APL and on the effective workspace size.

19.8 SHARP APL now issues a `□sc` post to all tasks suspended for a state-change wait when a shutdown of APL is pending. This permits tasks to prepare for the impending shutdown by performing internal housekeeping chores.

The shared variable functions `□svn`, `□svo`, and `□svq` are modified to permit SHARP APL Version 20 to be used in conjunction with I.P. Sharp's Network Shared Variable Processor (NSVP) which will be available sometime in the future.

Other Enhancements

The following changes to SHARP APL make the system easier to use, more versatile, and more efficient.

19.8 The fix functions ⎕*fx* and 3 ⎕*fd* are changed to simplify the importing of APL functions from other APL systems to SHARP APL.

The system editor for APL functions is changed to make it more versatile in the handling of error conditions.

Messages to the SHARP APL operator are now routed to the system console rather than to *opr*1 (account number 314159).

The scope of the character control system command,) *cc*, is changed to make the setting of the character control table easier by including default settings for individual accounts or for the entire system. The character set is expanded to permit easier translation between the available settings. When an account is logging on to the system, either character control setting is accepted.

The character control function is available to allow you to set the character control table within an application.

Products and Functions No Longer Supported

The following products and functions are not supported in Version 20 of SHARP APL:

The APL MPX interface (AMPX), which provided access to SHARP APL systems for asynchronous terminals, is replaced by the ATH/E terminal handler.

DCTTAPE and IBM 2741 terminals are no longer supported by SHARP APL.

This chapter contains information on the new and changed primitive functions available with SHARP APL. Formal definitions of some functions are reprinted from Kenneth Iverson's "Dictionary of APL."

In (⍷) Added

The dyadic primitive function ⍷ (**in**, also referred to as **string search** or **epsilon-underbar**) indicates all occurences of one array within another. For an array ⍵ and a pattern (array) ⍺, the statement

```
b←⍺⍷⍵
```

produces a Boolean array *b* such that the **ones** in *b* indicate the beginning points of the pattern ⍺ in ⍵. The array *b* has the same shape as ⍵. Both ⍺ and ⍵ can be arrays of any rank.

This function is formally defined as follows. Note that ⍨⍤3 (3 cut) is not yet implemented in SHARP APL.

```
⍺⍷⍵ ≡ ((⍳1,⍴⍺)⍨⍤3<⍵)∈<⍺
```

The following two examples illustrate the use of ⍷:

```
      'to' ⍷ 'toronto'
1 0 0 0 0 1 0


      (0 1-.+⍳3) ⍷ 4|i-.+i←⍳5
1 0 0 0 0
0 0 0 0 0
0 0 1 0 0
0 1 0 0 0
0 0 0 0 0
```

Nubsieve (≠) Added

The monadic primitive function ≠ (**nubsieve**) indicates the unique major cells within an array. For an array *x* of any rank, the statement

```
b←≠x
```

produces a Boolean vector *b* such that:

- the elements in *b* correspond to the major cells of *x*
- an element of *b* will be a **1** if its corresponding major cell is unique with respect to all preceding major cells; otherwise it is a **0**

For lists, nubsieve can be formally defined as follows (using Direct Definition notation). For arrays, major cells are compared in place of list elements.

```
ns: ⍴0: x⍴⍵: (<\~a)∨a\ns(a←⍵≠1⍴⍵)/⍵
```

Note that if the comparison tolerance variable (⎕*ct*) is not needed in the comparisons, this definition can be simplified to `ns: (⍵⍳⍵)=⍳⍴⍵`.

The following three examples illustrate the use of the nubsieve primitive:

```
    ≠'toronto'
1 1 1 0 1 0 0

    ≠(1 3⍴⍳3)⍪3 3⍴⍳9
1 0 1 1

    (⍪b)⊃x⊃(b←≠x)≠x←>¯1¨<' bill adam james bill
george ed james ed john'

1 bill   bill
1 adam   adam
1 james  james
0 bill   george
1 george ed
1 ed     john
0 james
0 ed
1 john
```

Format (⍕) Modified

The dyadic primitive function ⍕ (**format**, often referred to as **thorn** to distinguish it from ⎕*fmt*) was modified (Release 19.8). Dyadic ⍕ now accepts an empty vector as its left argument, causing the right argument to be formatted according to the rules of monadic ⍕. This can be stated as follows:

```
('' ⍕ ω) ≡ ⍕ω
```

The following example illustrates the use of dyadic ⍕:

```
      name←60 10⍴86↓⎕av ⍝ employee names
      pay←? 60 1⍴4000   ⍝ pay
      empno←? 60 1⍴1000 ⍝ employee number

      ∇r←old
[1]   r←'10a1,i5,f8.2' ⎕fmt name⊃empno⊃pay
      ∇

      ∇r←new;⎕ps
[1]   ⎕ps←0
[2]   r←⍕ ('' ⊃5 0⊃8 2)⍕⍤1⍤0⊢ name⊃empno⊃pay
      ∇
```

In addition to using the new formatting capabilities, the function *new* executes approximately three times faster than the function *old*, which uses ⎕*fmt*.

Raze (↓) Added

The monadic primitive function ↓ (**raze**) is used to reconstitute arrays of data by assembling the opened elements of the array along the leading axis. The array may be of any rank.

Raze is formally defined as follows. Note that the first and third parts of this definition are not valid statements in SHARP APL.

```
(↓ω) ≡ ⍪⍤1⍤>⊢⍬⍴ω          if the leading axis is greater than one

(↓ω) ≡ (1↓⍴ω)⍴<⍴0         if the leading axis equals zero

(↓ω) ≡ 1⍴⍤1⍤>(1↓⍴ω)⍴ω     if the leading axis equals one
```

This definition can be summarized as follows.

If the leading axis of the argument is greater than one, the result is computed by disclosing the scalar elements making up the argument (if necessary), catenating the elements along the leading axis, and enclosing the result. The rank of the result is one less than the rank of the argument.

For scalars and non-empty vectors, the result is the scalar or vector enclosed. (The elements making up the argument are disclosed, if necessary.)

For empty arrays (for example, `5 0 4 ⍴ ''`), the result is empty.

The following special cases are handled by ↓.

- For an array of shape `0 ⍴ v` (where *v* is any integer vector), the result has the shape *v*. If *v* is an empty array, the result is empty. If *v* is not empty, the result is computed as described above. This is shown in the following example (⎕*ps* is set to `¯1 ¯1 ¯2 ¯2`):

```
      ↓0 1 3 ⍴ ⍳3
|||||
|||||
|||||
```

- For arrays with a leading axis of one, the shape of the result is obtained by dropping the leading axis. (That is, for an array of shape `1 ⍴ n`, with *n* being any vector, the result has the shape *n* .) If the resulting array is not empty, the cells of the result are enclosed. If the resulting array is empty, the result is empty. This can be seen as follows:

```
      ⍴1 3 0 4 ⍴ 0
1 3 0 4
      ⍴↓1 3 0 4 ⍴ 0
3 0 4
```

The following additional example illustrates the use of monadic ↓. (For this example, ⎕*ps* is set to ¯1 ¯1 ¯2 ¯2.) In the example, a character string is cut into words, the word *and* is replaced with the word *or*, and the revised string is displayed.

```
      x←2⍤0<'This and that and these and those. '
┌─────┐┌───┐┌─────┐┌───┐┌──────┐┌───┐┌───────┐
│This ││and ││that ││and ││these ││and ││those. │
└─────┘└───┘└─────┘└───┘└──────┘└───┘└───────┘

      x[(x≡⍤0<'and ')/⍳⍴x]←<'or '

      x
┌─────┐┌──┐┌─────┐┌──┐┌──────┐┌──┐┌───────┐
│This ││or ││that ││or ││these ││or ││those. │
└─────┘└──┘└─────┘└──┘└──────┘└──┘└───────┘

      >↓x
This or that or these or those.
```

Disclose (>) Modified

The monadic primitive function > (**open** or **disclose**) has been extended to provide permissive treatment of the individual cells of a boxed array. This allows you to disclose several boxes at one time, with the data fitting together in a regular shape even when the boxes contain data of different shapes. When applied to an array with no boxed elements, disclose has no effect.

The rank of the disclose function is zero; the shape of its argument is also its frame. To construct an array within this frame, each major cell in the result of the disclose operation must share a common shape. This shape is based on the maximum rank and shape of the individual cells after the disclose operation.

If the ranks of items differ after the disclose operation, items of lesser rank are raised to the maximum common rank through the addition of leading unit lengths. If the shapes of items differ, items are brought into common shape by taking the maximum common shape and padding with the fill element. The shape of the result is the shape of the argument catenated to the maximum common shape. In the example below, the shape of the argument (4) is catenated to the maximum common shape after the disclose of the individual items (5), giving the result a shape of 4 5.

The following example illustrates the permissive disclose function. For this example □*ps* is set to ¯1 ¯1 ¯2 ¯2.

```
      ω←¯1⍕<' the quick brown fox.'
      ω
|---||-----||-----||----|
|the||quick||brown||fox.|
|___||_____||_____||____|

      >ω
the
quick
brown
fox.

      ⍴>ω
4 5
```

This chapter describes the additions and changes to the SHARP APL file system.

⎕*fhold* Modified

The function ⎕*fhold* has been modified to allow you to hold a range of file components rather than the entire file. This modification permits several users to hold the same file simultaneously as long as the ranges held do not overlap.

The format of the modified ⎕*fhold* function is as follows:

```
r←⎕fhold ⍵
```

where ⍵ specifies the arguments for the file hold. The range of file components to be held is specified in ⍵ as follows. In all cases, the range is considered empty if the lower bound is greater than the upper bound.

- If ⍵ is an array with four rows, for any given column the first row is the tie number, the second row is the pass number, the third row is the lower bound, and the fourth row is the upper bound of the range. Bounds are specified as integers from -2147483648 to 2147483647, inclusive.

- If ⍵ is an array with three rows, the first row is the tie number, the second row is the pass number, and the third row is an integer from -2147483648 to 2147483647, inclusive, specifying the component to be held.

- If ⍵ is a scalar, a vector, or an array with one or two rows, the range is assumed to be from -2147483648 to 2147483647, inclusive.

Executing the monadic function ⎕*fhold* (or ⎕*hold*) releases the held ranges of all currently held files **and then** acquires holds on the specified ranges.

Executing the dyadic function `1 ⎕fhold` retains the held ranges on all currently held files and acquires holds on the specified ranges. If a file is currently held and is specified in the argument to `1 ⎕fhold`, the new range must be identical to the old range (or a *domain error* results). When `1 ⎕fhold` is issued using a three or four row right argument, it results in a three row array; each column of this array contains the tie number (first row), lower bound (second row), and upper bound (third row) of a held range.

Library Access Controls Added

With SHARP APL Version 20, an access control matrix can be associated with an entire file library, as well as with individual files. This library access matrix, which is similar in structure to the matrices used to grant access to files, is set by the ⎕*stacl* function (described later in this chapter). When it is set, the library access matrix defines the following:

- which accounts can create and rename files in the library
- which file names are returned from a ⎕*lib* request
- which accounts can read and set the library access matrix.

The format of the library access matrix is an *n* by 3 matrix. As with file access matrices, each row of the matrix contains the account number, permission number, and pass number.

Account number. This is the APL account number to which you are assigning permission. A zero (0) in this column grants the access permission specified to all accounts that are not explicitly specified elsewhere in the access matrix for the given pass number.

Permission number. This number defines the permission granted to the specified account. A permission number of minus one (¯1) grants unlimited access to the library. Other permission numbers are the sums of the appropriate permission codes in the following table. (For example, to grant permission to rename files and to read the library access matrix, set the permission number to '18' -- code 2 plus code 16.)

Code	Operation permitted
1	Create files in library (⎕*create*)
2	Rename files in library (⎕*rename*)
4	List files in library (using ⎕*lib* from another account) -- if this code is not specified, no files are listed; if it is specified, only files that the account has some form of access to are listed
8	List all files in library (using ⎕*lib* from another account); note that an account must also have permission code '4' to list another account's files
16	Read the library access matrix (⎕*rdacl*)
32	Set the library access matrix (⎕*stacl*)

Pass number. This number is the pass number needed for the specified access. A zero (0) in this column permits the access specifed to the appropriate accounts without a pass number or with the pass number '0'.

A typical library access matrix might look like the following:

```
1111112        ¯1          0
1726354         3        987
      0        12        654
```

The library access granted by this matrix is as follows. Note that this access is in addition to the access implicitly associated with each library access matrix (described later in this section).

- The first row (`1111112  ¯1  0`) permits account number `1111112` unlimited access with no pass number to the library files.

- The second row (`1726354  3  987`) permits account number `1726354` to create and rename files in the library using pass number `987`.

- The third row (`0  12  654`) permits any account to list all of the files in the library using pass number `654`, regardless of individual file access.

Three rows of default access information are implicitly associated with each library access matrix. These rows are as follows:

```
library        ¯1          0
      0         4          0
      0         0    (not 0)
```

The first row (*`library`* `¯1  0`) ensures that an account always has complete access to its own library. This access information takes precedence over the information in the access matrix set by □*`stacl`*.

The second and third rows take effect **after** the information in the access matrix set using □*`stacl`* and provide default access for accounts not referred to in the matrix.

The second row (`0  4  0`) sets the permission code to `'4'` for all accounts not specified (either explicitly or implicitly using account number 0) in the matrix assigned by □*`stacl`*. This permits accounts to list only those files in the library that the account has some form of access to, unless additional permission is specified elsewhere in the library access matrix.

The third row (`0  0  (not 0)`) sets the permission code to 0 (zero) for all non-zero library pass numbers not explicitly specified using □*`stacl`*.

File functions that are affected by the library access matrix are: □*`create`*, □*`rename`*, □*`lib`*, □*`rdacl`*, and □*`stacl`*.

⎕create

The ⎕create function now accepts a library pass number in its right argument. The format of the ⎕create function is as follows:

```
'fn' ⎕create tn (,cn (,lpn))
```

where *fn* is the name of the file being created, *tn* is the tie number, *cn* is the component number, and *lpn* is the library pass number. (The concept and use of the component number is described later in this chapter.)

⎕rename

The ⎕rename function now accepts a library pass number in its right argument. The format of the ⎕rename function is as follows:

```
'fn' ⎕rename tn (,pn (,lpn))
```

where *fn* is the name of the file being renamed, *tn* is the tie number, *pn* is the file pass number, and *lpn* is the library pass number.

⎕lib

The ⎕lib function now accepts a library pass number in its right argument. The format of the ⎕lib function is as follows:

```
⎕lib library (,lpn)
```

where *library* is the library number and *lpn* is the library pass number.

⎕rdacl

A new function, ⎕rdacl, allows certain accounts to read the library access matrix for a specified library. The format of the ⎕rdacl function is as follows:

```
⎕rdacl library (,lpn)
```

where *library* is the library number and *lpn* is the library pass number.

⎕stacl

A new function, ⎕stacl, allows certain accounts to set the access matrix of the specified library. The format of the ⎕stacl function is as follows:

```
lam ⎕stacl library (,lpn)
```

where *lam* is the library access matrix (described earlier in this chapter), *library* is the library number, and *lpn* is the library pass number.

⎕create Modified

The ⎕create file function now accepts the starting component number as a right argument. The format of the ⎕create function is as follows:

```
'fn' ⎕create tn (,cn (,lpn))
```

where *fn* is the name of the file being created, *tn* is the tie number, *cn* is the starting (and next to be used) component number, and *lpn* is the library pass number. The component number must be a positive integer. If a library pass number is specified, the starting component number must be supplied and must be non-zero.

The following is an example of the ⎕create command. In this example, a file called *wanda* is created starting at component number 68. (The ⎕size function verifies the creation and starting component number of the file.)

```
      'wanda' ⎕create 1 68
      ⎕size 1
68 68 0 102620
```

⎕rdfi Added

A new system function that reports on file information, ⎕rdfi, is now available. This function provides the account number and time stamp information associated with the following file operations:

- the creation of the file
- the latest setting of the file access matrix
- the latest alteration of the file.

The permission code for using ⎕rdfi is 65536.

The format of the ⎕rdfi function is as follows:

```
z←⎕rdfi tn (,pn)
```

where *tn* represents the file tie number and *pn* represents the file pass number, which is optional.

The result of □*rdfi* is a matrix of shape 4 2. This matrix contains the following information. The time stamp information is given in sixtieths of a second since March 1, 1960. The function *ftt* in the *1 ts* workspace can convert this to □*ts* format.

- The first row of information relates to the creation of the file (using the □*create* function). It contains the number of the account that created the file followed by the time stamp information from when the file was created.

- The second row relates to the latest setting of the file access matrix (using the □*stac* function). It contains the number of the account that last set the file access matrix followed by the time stamp information from when it was set.

- The third row is reserved for future use. It is currently set to minus ones (¯1's).

- The fourth row relates to the latest alteration of the file using one of the following functions: □*append*, □*appendr*, □*drop*, □*rename*, □*replace*, or □*resize*. It contains the number of the account that last altered the file followed by the time stamp information associated with that alteration.

The following is a sample result of the □*rdfi* function.

```
 1.726354000e6          5.580219021e10
 1.726354000e6          5.589528844e10
¯1.000000000e0         ¯1.000000000e0
 7.858878000e6          5.597212603e10
```

Note: The number of rows of information provided by □*rdfi* may increase with future releases of SHARP APL. Avoid using the statement

```
¯1↑,□rdfi tn
```

to obtain the time of the latest alteration of a file, as it may provide incorrect results in future releases of SHARP APL.

CHAPTER 4. SYSTEM ENHANCEMENTS

This chapter describes the new and changed functions available to set or report on system information.

Transaction Reporting Facility Added

The Transaction Reporting Facility monitors the use of SHARP APL at the transaction level. Using this facility, you can specify which events you wish to record, such as calls to a database or time elapsed while using specific applications.

By developing applications based on the information collected by this facility, you can implement a wide variety of projects based on transaction data. These could include such projects as usage statistics, application performance analyses, and customer billing systems.

For information on the design and use of the Transaction Reporting Facility, refer to the *Transaction Reporting Facility Programmer's Guide*, publication code 1054-8909-E20.

Function Monitor Facility (□*fm*) Added

The system function □*fm* (the Function Monitor Facility) measures and reports on the CPU time and elapsed time used during the execution of user-defined functions (Release 19.8).

□*fm* can be used to monitor user-defined functions as well as individual lines within functions.

Using □*fm*, you can specify which workspace functions are to be monitored and the amount of detail you want reported. For example, you can monitor the number of times a function is executed and the amount of CPU time and elapsed time spent during execution (with or without monitoring subfunction calls). You can monitor individual lines within a workspace function, reporting on the number of times the lines were executed, CPU time, elapsed time, and so on.

For more information on □*fm*, refer to the *Function Monitor Facility User Guide*, publication code 0749-8711-E1.

Termination Workspace ID (⎕*twsid*) Added

The active workspace of a task that was interrupted by an external event is saved by SHARP APL whenever possible. For terminal tasks and shared tasks (T-tasks and S-tasks), the name of the saved workspace is *continue*. For batch tasks and non-terminal tasks (B-tasks and N-tasks), the name is determined from the parameters set with ⎕*run*.

A new system function, ⎕*twsid*, is available. Using this function, you can perform the following:

- Inquire whether the active workspace is to be saved if the task terminates abnormally.
- Specify that the active workspace is not to be saved if the task terminates abnormally.
- Specify the workspace ID to be used for saving the active workspace if the task terminates abnormally. This workspace ID can include a lock on the workspace or a library in which to store the workspace. (The library must belong to the user number that the task was run on.)

The format of the ⎕*twsid* function is as follows:

```
⎕twsid α
```

where α is one of the following:

1 (one)	Save the active workspace using the current termination workspace ID
¯1 (minus one)	Do not save the workspace
0 (zero)	Return the current setting of ⎕*twsid* (one or minus one)
'' (empty vector)	Return the current termination workspace ID
'*string*'	Set the termination workspace ID to the identifier specified in *string*. This may be one of the following: - A valid workspace name (such as '*crashid*') - A valid workspace name followed by a colon and a password (such as '*crashid:lock*') - The account's library number followed by a space, a valid workspace name, and (optionally) a colon and password (such as '1726354 *crashid:lock*')

The termination workspace ID can also be defined in terms of information that is associated with the current task. For example, the statement

```
⎕twsid 'abc',⍕1⍴⎕runs
```

sets the workspace termination ID to ABCxxxx, with xxxx being the task ID.

Notes

1. To save a termination workspace with a name other than *continue*, the account workspace quota must allow the account to save at least one more workspace.

2. The result of ⎕*twsid* with a valid numeric right argument is the previous setting of the numeric value of ⎕*twsid*.

3. The result of ⎕*twsid* with either an empty vector or a valid identifier as its right argument is a 22 character vector containing the library number and identifier of the previous setting of the termination workspace ID. The password, if any, is not returned.

2 ⎕*ws* 3 Modified

The workspace reporting function 2 ⎕*ws* 3 now provides information on the pending shutdown of APL and on effective workspace size.

2 ⎕*ws* 3 results in a 12-element vector. The second element of this vector, (2 ⎕*ws* 3)[⎕*io*+1], indicates pending shutdown of APL. Its value is normally 0 (zero). If its value changes from 0 to 1 (one), a shutdown is pending.

When a shutdown is started, new tasks may not be initiated on the system. For existing tasks, some parts of APL may be withdrawn prior to the interruption of user tasks (such as HCPRINT or the batch task scheduler).

You may want your batch applications to check for pending shutdown at certain points to permit general housekeeping tasks to be performed before the shutdown. The ability to detect and prepare for a shutdown can be used to ease the recovery and integrity, checking problems that normally result from a system crash. For more information on this topic, refer to "⎕*sc* Post at Shutdown," later in this chapter.

The third element of the 12-element vector result of 2 ⎕*ws* 3 provides the effective size for a clear workspace. This element, (2 ⎕*ws* 3)[⎕*io*+2], reports the actual size in bytes of the workspace that is under direct control of the APL application, excluding control blocks internal to the interpreter.

The ability to determine actual workspace size can be useful when you are designing large applications. For example, since you can now accurately determine the amount of area available for demand-paging space, you may want to consider implementing a demand-paging scheme for APL objects within the workspace.

6 ⎕*ws* Modified

The result of issuing 6 ⎕*ws* with the name of a shared variable with no value as a right argument is now a *domain error*. In previous releases of SHARP APL, the result was a *result error*.

⎕*sc* Post at Shutdown

An impending shutdown of APL can now be ascertained by examining the second element of 2 ⎕*ws* 3 (see "2 ⎕*ws* 3 Modified," earlier in this chapter). Using this information, tasks can perform internal housekeeping chores before the shutdown occurs (assuming that, for normal shutdowns, your installation has a period of grace in which tasks can complete before they are interrupted).

When an APL shutdown has been initiated, all tasks suspended for a state-change wait (a ⎕*sc* wait, not a shared variable wait) are posted. This post can benefit applications that employ background tasks or server tasks. Consider the following generalized function:

```
     ∇ serve
[1] go:process                          ⍝ work to do
[2]  →(2 ⎕ws 3)[⎕io+1]⍴shutdown         ⍝ is shutdown pending?
[3]  →⎕sc⍴go                            ⍝ wait for some work
[4] shutdown:cleanup                    ⍝ general cleanup
     ∇
```

In the event of a shutdown, the function does not immediately suspend on execution of ⎕*sc*; it continues as if a post had just occurred. This gives programs the opportunity to detect and prepare for shutdown. This should streamline recovery procedures when APL restarts. For example, if the application writes a "clean shutdown" record when it ends normally, checking for this component would determine whether you need to perform the recovery and integrity checking procedures normally undertaken after abnormal termination.

Changes to Shared Variable Functions

Changes to the shared variable function ⎕*svn*, ⎕*svo*, and ⎕*svq* are included in SHARP APL Version 20. These changes permit Version 20 to be compatible with I.P. Sharp's Network Shared Variable Processor (referred to as NSVP) once it is released for distribution.

Monadic ⎕*svn* Modified

The monadic ⎕*svn* function is now able to accept a wider range of right arguments. Possible values for the right argument now include character vectors and three-element arrays of character vectors. (Originally, the function only accepted a numeric scalar as a right argument.)

The format of monadic ⎕*svn* is as follows:

```
⎕svn arg
```

where *arg* is a numeric scalar, an enclosed scalar or vector, or a character vector.

If *arg* is a scalar and your active workspace is not sharing variables using a different clone ID, ⎕*svn* sets the clone ID for your active workspace to *arg*. The result is the current clone ID (either the value of *arg* or the previously existing clone ID). If your active workspace does not have a current clone ID and the requested clone ID cannot be implemented because another task with the same processor ID is using the specified clone ID, the result is ¯1.

If *arg* is a character vector representing the system ID, the result is the numeric value of the system ID. The result is ¯1 if your workspace does not currently have a clone ID and the default clone ID is being used by another task with the same processor ID. (The default clone ID is 0 (zero).)

If *arg* is a three-element array of character vectors (containing the system ID, account name, and account password, respectively), the result is the numeric value of the system ID or ¯1.

The following examples show the use of ⎕*svn*:

```
      ⎕svn 'mvs05'
305050

      ⎕svn 'sys1'⊃'gibson'⊃'sg1962'
301010
```

Note: Your acitve workspace can only use one numeric system ID at a time. To change the numeric system ID, you must end the link to the NSVP on the remote system using dyadic $\square svn$, then reassign the system ID using monadic $\square svn$.

Dyadic $\square svn$ Added

Dyadic $\square svn$ is now available. It is used to disconnect the link to the NSVP on the remote system. The format of dyadic $\square svn$ is as follows:

```
nnn  □svn ''
```

where *nnn* is either the numeric system ID assigned by monadic $\square svn$ or 0 (zero). If it is a system ID, the specified link is disconnected. If it is zero, any existing link to the issuing account is disconnected.

$\square svq$ Modified

The $\square svq$ function now accepts a wider range of right arguments. Possible values for its right argument now include an enclosed null value and a three-element vector. (Originally, it only accepted a null value and a two-element vector for its right argument.)

If the right argument is an enclosed null value (for example, `<⍳0`), the result is a three-column matrix, with each row containing the system ID, processor ID, and clone ID of an outstanding offer to share a variable.

If the right argument is a three-element vector containing system ID, processor ID, and clone ID, the result is a matrix with each row containing the name of a variable the specified processor is offering to share with your account.

The following examples illustrate the use of $\square svq$. Note the `¯1` in the result of the first example. This shows that the offer was made by a task on the same system as your active workspace (not on a remote system). You can use `¯1` as the system ID in a three-element vector argument to $\square svq$ or in a three-column matrix argument to $\square svo$; this treats variables shared with tasks on your system in the same manner as variables shared with tasks on a remote system.

```
      □svq <⍳0
304050  314158     99
    ¯1     124   4017

      □svq (304050 314158 99)
var01
apple2
revnum9
```

□*svo* Modified

Dyadic □*svo* now accepts a three-column matrix as a left argument. The three columns should contain the system ID, processor ID, and clone ID of task or tasks to which you offered the variable(s) specified in the right argument. The result of this function is the degree of coupling for the specified variables. Note that if you specify a system ID of ¯1, you are offering to share variables with a task (represented by the processor ID and clone ID) on the same system as your active workspace.

The following example shows the use of □*svo*:

```
      (1 3 ⍴ 304050 314158 99) □svo 'var01'
2
```

This chapter describes other changes and additions to SHARP APL.

Fix Functions ⎕*fx* and 3 ⎕*fd* Modified

The fix functions ⎕*fx* and 3 ⎕*fd* have been modified to simplify the importing of APL functions from other APL systems to SHARP APL (Release 19.8).

The fix functions no longer produce an error result in the following situations:

- for empty (all blank) lines
- for lines containing the following invalid expressions:
  - unbalanced quotes
  - third alphabet characters (⎕*av*[166+⍳26]) not in quoted strings
  - invalid system function names or system variable names
  - invalid numeric constants.
- for lines containing the box-drawing characters for IBM 3270-type terminals (⎕*av*[241+⍳11]) in quoted strings passed to the fix function.

System Function Editor Modified

The system function editor (∇ editor) has been modified (Release 19.8) as follows:

- Empty (all blank) lines created by one of the fix functions can be displayed or deleted. Empty lines cannot be entered using the system function editor.
- You can enter lines containing the following invalid expressions without producing an error result:
  - unbalanced quotes
  - third alphabet characters (⎕*av*[166+⍳26]) not in quoted strings
  - invalid system function names or system variable names
  - invalid numeric constants.

- You can display and delete lines containing the box-drawing characters for IBM 3270-type terminals (`⎕av[241+⍳11]`). Lines containing these characters cannot be entered or edited using the system function editor.

`)opr` and `)oprn` Messages to System Console

The system commands `)opr` and `)oprn`, used for sending messages to the SHARP APL operator, no longer send messages to `opr1` (account number `314159`). These commands now send messages to the system console.

If you require a reply from the operator, use the `)opr` command; your messaged is highlighted at the system console until it is answered. Your keyboard locks and you suspend execution until the reply is recieved.

Notes:

1. If you send a message to `)opr` and then break out of the wait state, it is unlikely that you will receive a reply to your message from the operator.

2. Since the `)opr` and `)oprn` commands now send messages to the system console, sending "polite" messages (such as "`thanks opr.../rick`") to the APL operator should be avoided to help keep the system console free for important messages.

Modified Character Control for Non-APL Asynchronous Terminals

Several changes have been made to the character control system command, `)cc`. (For a description of this command, refer to the *Release 19 Guide for APL Programmers*, publication code 0375-8703-E19.)

The following changes to `)cc` might affect your application programs:

- The character control default setting (`new` or `old`) for your SHARP APL system can now be set when the system is started up.

- The character control default setting for a particular account number can be set in the user profile.

- Valid character control settings now include `normal` (the same as `new`) and `reverse` (the same as `old`).

- Response from the command is either `normal` (if the previous setting was `normal` or `new`) or `reverse` (if the previous setting was `old` or `reverse`).

- Additional characters have been included in the character set of *old* so that data written for *new* translates correctly when *old* is specified. These are the characters %, ", |, @, _, `, &, and # and the box-drawing characters for IBM 3270-type terminals (⎕*av*[240+⍳11]).

  Future enhancements to SHARP APL will permit the ATH/V and ATH/E terminal handlers to support these characters.

Non-APL Asynchronous Terminal Character Control Function

A new monadic character control function, *cc*, is available (Release 19.8). This function is similar to the system command) *cc*. (For more information on the) *cc* command, refer to the *Release 19 Guide for APL Programmers*, publication code 0375-8703-E19 and "Modified Character Control for Non-APL Asynchronous Terminals" in this chapter.) Both *cc* and) *cc* provide enhanced support for asynchronous terminals, but *cc* can be used in user-defined functions, unlike the) *cc* command.

The function is available in the 1 *wsfns* workspace. The format of the function is as follows:

```
z←cc α
```

If α is an empty vector (' '), the result is a character vector containing the current translation table name.

If α is a character vector with a maximum of 7 characters (including leading or trailing blanks) containing the name of a valid translation table, the current table is changed to the specified name and the result is a character vector containing either *normal* or *reverse*. It is *normal* if the previous setting was *new* or *normal*; it is *reverse* if the previous setting was *old* or *reverse*.

If α is an invalid argument (such as an unsupported table name), the result is a *domain error*. If α is a character vector longer than seven characters, the result is a *length error*.

There are two valid settings for *cc*: *old* or *reverse* and *new* or *normal*. The results of these settings are described in the *Release 19 Guide for APL Programmers*.

Note: If the function *cc* is already used in an application, the translation table can also be selected using an alternate function, *Cc*, also in the 1 *wsfns* workspace.

This chapter lists the products and functions not supported by SHARP APL Version 20.

APL MPX Interface (AMPX) Replaced

The APL MPX Interface (AMPX), which provided access to the SHARP APL system through asynchronous terminals, is no longer supported. Some terminal types which used this interface, such as IBM 2741-type terminals, are no longer supported. Asynchronous ASCII terminal support is now available using the Asynchronous Terminal Handler/Emulation Program Support (ATH/E).

For more information on asynchronous ASCII terminal support, refer to the *EPH User's Guide*, publication code 0738-8703-E1.

Terminal Types That Are No Longer Supported

SHARP APL no longer supports the following terminal types:

- DCT500 terminals operating at 300 baud
- IBM 2741-type terminals

For a list of asynchronous terminals currently supported, load the `5 term` workspace and display the variable `terminals`.

AMPX (the APL MPX interface) replaced 29
APL character set, representation of iii - iv
asynchronous terminal support 29

cc function. *See* character control function.
character control function (*cc*) 27 - 28
character control system command ( ) *cc*) 27

DCCTAPE terminal 3
DCT500 terminal 29
disclose function (>) 9

epsilon-underbar function. *See* in function.

first alphabet. *See* APL character set.
fix functions
  □*fx* 25
  3 □*fd* 25
format function (⍕) 7
ftt function to convert dates to □*ts* format 16
Function Monitor Facility (□*fm*) 17

holding files 11

in function (⍷) 5
index origin system variable (□*io*) iii

library access matrix
  definition of 12
  format of 12 - 13
  functions affected 13
  permission codes, list of 12
  reading the matrix. *See* □*rdacl*.
  setting the matrix. *See* □*stacl*.

messages to the APL operator 26

nubsieve function (≠)
  description 6
  without comparison tolerance function 6

open function. *See* disclose function.

position and spacing variable (⎕*ps*) iii

⎕*av*, location of letters in iv
⎕*create* 14, 15
⎕*ct*, used with nubsieve function 6
⎕*fhold* 11
⎕*fm* 17
⎕*fx* 25
⎕*hold* 11
⎕*io* iii
⎕*lib* 14
⎕*ps* iii
⎕*rdacl* 14
⎕*rdfi* 15 - 16
⎕*rename* 14
⎕*sc* post at shutdown 20
⎕*stacl* 14
⎕*svn*
  dyadic 22
  monadic 21
⎕*svo* 23
⎕*svq* 22
⎕*twsid* 18 - 19

raze function (↓) 7 - 9
read file information function (⎕*rdfi*)
  format 15
  functions affected 16
  permission code required 15
  result of 16
releasing a file hold 11

second alphabet. *See* APL character set.
shared variable functions
  ⎕*svn* 21, 22
  ⎕*svo* 23
  ⎕*svq* 22
  6 ⎕*ws* function 20
string search function. *See* in function.
system function editor (∇ editor) 25

terminal types that are no longer supported 29
termination workspace ID function (⎕*twsid*) 18 - 19
third alphabet. *See* APL character set.
thorn function. *See* format function.

Transaction Reporting Facility 17

1 □*fhold* 11
2741-type terminals 29
2 □*ws* 3 function 19
3 □*fd* function 25

) *cc* system command 27
) *opr* system command 26
) *oprn* system command 26

∇ editor 25
⍒ function 7
⊆ function 5
≠ function 6
> function 9
↓ function 7